Strong Equivalence of Logic Programs under the Infinite-Valued Semantics

Christos Nomikos
Department of Computer Science, University of Ioannina
P.O. Box 1186, 45110 Ioannina, Greece

Panos Rondogiannis
Department of Informatics & Telecommunications, University of Athens
Panepistimiopolis, 157 84 Athens, Greece

William W. Wadge
Department of Computer Science, University of Victoria
PO Box 3055, STN CSC, Victoria, BC, Canada V8W 3P6

Abstract

We consider the notion of strong equivalence [4] of normal propositional logic programs under the infinite-valued semantics [7] (which is a purely model-theoretic semantics that is compatible with the well-founded one). We demonstrate that two such programs are strongly equivalent under the infinite-valued semantics if and only if they are logically equivalent in the infinite-valued logic of [7]. In particular, we show that strong equivalence of normal propositional logic programs is decidable, and more specifically **coNP**-complete. Our results have a direct implication for the well-founded semantics since, as we demonstrate, if two programs are strongly equivalent under the infinite-valued semantics, then they are also strongly equivalent under the well-founded semantics.

Keywords: Formal Semantics, Negation in Logic Programming, Strong Equivalence.

1 Introduction

The notion of strong equivalence of logic programs was introduced in [4]. Strong equivalence is a very intuitive notion, which appears to have direct consequences in the area of program transformations. Two logic programs P_1 and P_2 are termed strongly equivalent under a given semantics if for all logic programs P, $P_1 \cup P$ has the same meaning as $P_2 \cup P$ under this given semantics. Obviously, when two logic programs are strongly equivalent, we can replace one for the other inside a bigger program without any change in the observable behavior of this program. In [4] it is demonstrated that two programs are strongly equivalent under the answer set semantics if and only if they are equivalent in the logic of here-and-there (HT). The significance of HT in the theory of logic programming was initially uncovered by the results of D. Pearce in [6] and strengthened by the results regarding strong equivalence in [4].

The answer set semantics is one of the two main streams of research with respect to negation in logic programming. The other stream is what is usually termed the "*canonical model semantics*", which seeks among the models of the program a unique "intended model". Recently, two of the authors obtained a purely model-theoretic canonical model semantics for

logic programs with negation, termed the *infinite-valued* semantics [7]. This new semantics is compatible with the well-founded one [8], but it has the advantage of being purely logical: the meaning of a program is the unique minimum infinite-valued model with respect to an ordering relation that is independent from the syntax of the program. This new approach appears to be a useful tool in semantic investigations of logic programming. For example, the infinite-valued semantics has recently motivated a novel, purely game-theoretic approach to the semantics of negation in logic programming [2].

In this paper we consider the notion of strong equivalence of normal propositional logic programs under the infinite-valued semantics and we derive a simple and logical characterization: *two programs are strongly equivalent under the infinite-valued semantics if and only if they are logically equivalent in the infinite-valued logic of [7].* Moreover, although the underlying logic is based on a truth domain with an infinite number of truth values, we demonstrate that strong equivalence of normal propositional logic programs is decidable; more specifically, we prove that it is a **coNP**-complete problem. Finally, our results about strong equivalence under the infinite-valued semantics, directly provide a sufficient condition for strong equivalence under the well-founded semantics. A preliminary version of this article has appeared as a poster short-paper in [5]. In particular, the present article extends the previous one by providing detailed motivation and background, as well as proofs for all the results. Moreover, it extends the previous paper by Proposition 2.3 showing that for definite Horn programs strong equivalence coincides with classical logical equivalence.

The rest of the paper is organized as follows: Section 2 motivates the notion of strong equivalence and introduces its applicability beyond the answer set semantics. Section 3 introduces the infinite-valued semantics and its underlying infinite-valued logic. Section 4 demonstrates that under the infinite-valued semantics, strong equivalence of normal propositional logic programs coincides with logical equivalence. Section 5 examines the complexity of strong equivalence under the infinite-valued semantics. Finally, section 6 concludes the paper with relevant discussion.

2 Strong Equivalence of Logic Programs

In this paper we consider *normal propositional logic programs* built using propositional variables from a fixed set Q. Formally:

Definition 2.1 *A* normal propositional logic program *is a finite set of* normal program clauses *of the form:*

$$p \leftarrow q_1, \ldots, q_n, {\sim}r_1, \ldots, {\sim}r_m$$

where the q_i's, the r_i's and p are propositional variables from Q. A program is called definite *if in every clause of the above form, $m = 0$.*

The notion of *strong equivalence* was initially defined and used under the answer set semantics. However, it can be directly generalized to apply to other semantics:

Definition 2.2 *Two logic programs P_1 and P_2 are termed* strongly equivalent *under a given semantics if for every logic program P, $P_1 \cup P$ and $P_2 \cup P$ have the same meaning under this given semantics.*

Consider applying the above definition to definite logic programs using the standard minimum model semantics of logic programming. The following proposition gives an easy characterization for this case:

Proposition 2.3 *Two definite propositional logic programs P_1, P_2 are strongly equivalent under the minimum model semantics if and only if P_1 is logically equivalent to P_2.*

PROOF. ($\Leftarrow$) Assume that P_1 and P_2 are logically equivalent. Then, $P_1 \cup P$ and $P_2 \cup P$ are also logically equivalent, i.e., they have the same sets of models, which implies that $P_1 \cup P$ and $P_2 \cup P$ have the same minimum model (which exists by the model intersection theorem).

($\Rightarrow$) Assume on the other hand that P_1 and P_2 are strongly equivalent under the minimum model semantics, but not logically equivalent. Then, without loss of generality, we may assume that P_1 has a model M that is not a model of P_2. We construct a program P as follows: for each propositional symbol p in P_1 such that $M(p) = True$, P contains the clause $p \leftarrow$. Then, M is the minimum model of $P_1 \cup P$, since any proper subset of M would not satisfy some clauses in P. But then, M is also the minimum model of $P_2 \cup P$ since P_1 and P_2 are strongly equivalent. Therefore, M is a model of $P_2 \cup P$, which implies that M is a model of P_2 (contradiction). ∎

The above proposition provides an easy characterization of strong equivalence of negation-free logic programs under the minimum model semantics. However, things are much more complicated when one extends logic programming with negation-as-failure. The problems can be illustrated by a simple example. Consider the two programs $\{p \leftarrow\sim r\}$ and $\{r \leftarrow\sim p\}$. These programs are not strongly equivalent under any of the well-known semantics for negation. Moreover, notice that the two programs are logically equivalent when viewed as formulas of classical logic. In other words, it seems that one can not use classical logic (at least in any obvious way) in order to conclude that the above two programs are not strongly equivalent. These observations were originally presented in [4] for the case of the answer set semantics (where it was also demonstrated that strong equivalence in this case can be decided by checking for logical equivalence in the logic of here-and-there).

In the rest of this paper we examine strong equivalence under the infinite-valued semantics of [7]. The main theorem we obtain is very similar in nature to Proposition 2.3: two normal propositional logic programs are strongly equivalent under the infinite-valued semantics if and only if they are equivalent in infinite-valued logic. The similarity of these two results appears to suggest that the infinite-valued approach is of a fundamental importance in the study of the semantics of negation in logic programming.

3 The Infinite-Valued Semantics

In this section we introduce the basic concepts from [7] that we will need in this paper. The basic idea behind the infinite-valued semantics is that, in order to have a purely model theoretic semantics for negation-as-failure, one should use a richer logical framework than classical logic or even 3-valued logic. Informally, we consider an extended truth domain and use these extra values to distinguish between ordinary negation and negation-as-failure. Consider for example the program:

$$\begin{array}{lll} p & \leftarrow & \\ r & \leftarrow & \sim p \\ s & \leftarrow & \sim q \end{array}$$

Under the negation-as-failure approach both p and s receive the value *True*. We would argue, however, that in some sense p is "truer" than s. Namely, p is true because there is a clause which says so, whereas s is true only because we are never obliged to make q true. In a sense, s is true only by default. Our truth domain adds a "default" truth value T_1 just below the "absolute" truth value T_0, and a weaker false value F_1 just above ("not as false as") the absolute false value F_0. We can then understand negation-as-failure as combining ordinary negation with a weakening. Thus, $\sim \mathrm{F}_0 = \mathrm{T}_1$ and $\sim \mathrm{T}_0 = \mathrm{F}_1$. Since negation can effectively be iterated, our domain requires a whole sequence $\ldots, \mathrm{T}_3, \mathrm{T}_2, \mathrm{T}_1$ of weaker and weaker truth values below T_0 but above a neutral value 0 and a mirror image sequence $\mathrm{F}_1, \mathrm{F}_2, \mathrm{F}_3, \ldots$ above F_0 and below 0.[1] In [7] it is shown that, over this extended domain, every logic program with negation has a unique *minimum* model, under an appropriate ordering (see Definition 3.5 and Theorem 3.6 below); for the above program, this model is $\{(p, \mathrm{T}_0), (q, \mathrm{F}_0), (s, \mathrm{T}_1), (r, \mathrm{F}_1)\}$. Moreover, if in this model we collapse all the T_k and F_k to *True* and *False* respectively, we get the 3-valued well-founded model [8] of the program.

The infinite-valued semantics is therefore based on a truth domain with the ordering:

$$\mathrm{F}_0 < \mathrm{F}_1 < \mathrm{F}_2 \cdots < 0 < \cdots < \mathrm{T}_2 < \mathrm{T}_1 < \mathrm{T}_0$$

Intuitively, F_0 and T_0 are the classical *False* and *True* values and 0 is the *undefined* value. The intuition behind the new values is that they express different levels of truth and falsity. In the following we denote by V the set consisting of the above truth values. Interpretations are now defined as follows:

Definition 3.1 *An infinite-valued interpretation I is a function from the set Q of propositional symbols to the set V of truth values; I is extended to apply to literals, as follows:*

$$I(\sim q) = \begin{cases} \mathrm{T}_{n+1} & \text{if } I(q) = \mathrm{F}_n \\ \mathrm{F}_{n+1} & \text{if } I(q) = \mathrm{T}_n \\ 0 & \text{if } I(q) = 0 \end{cases}$$

for all $q \in Q$.

Satisfiability of a clause can now be defined:

Definition 3.2 *Let P be a program and let I be an infinite-valued interpretation. Then, I* satisfies *a clause $p \leftarrow \ell_1, \ldots, \ell_n$ of P if $I(p) \geq \min\{I(\ell_1), \ldots, I(\ell_n)\}$. Moreover, I is a* model *of P if I satisfies all clauses of P.*

Given an interpretation of a program, we adopt a specific notation for the set of atoms of the program that are assigned a specific truth value:

Definition 3.3 *Let P be a program, let I be an infinite-valued interpretation and let $v \in V$. Then, we define $I \parallel v = \{p \in Q \mid I(p) = v\}$.*

The following relations on infinite-valued interpretations will be needed:

[1] In fact, in [7] a T_α and a F_α are introduced for all countable ordinals α; since in this paper we deal with finite propositional programs, we will not need this generality here.

Definition 3.4 *Let I and J be infinite-valued interpretations and $n \in \mathbb{N}$. We write $I =_n J$, if for all $k \leq n$, $I \parallel \mathrm{T}_k = J \parallel \mathrm{T}_k$ and $I \parallel \mathrm{F}_k = J \parallel \mathrm{F}_k$. We write $I \sqsubset_n J$, if for all $k < n$, $I =_k J$ and either $I \parallel \mathrm{T}_n \subset J \parallel \mathrm{T}_n$ and $I \parallel \mathrm{F}_n \supseteq J \parallel \mathrm{F}_n$, or $I \parallel \mathrm{T}_n \subseteq J \parallel \mathrm{T}_n$ and $I \parallel \mathrm{F}_n \supset J \parallel \mathrm{F}_n$. We write $I \sqsubseteq_n J$ if $I =_n J$ or $I \sqsubset_n J$.*

Definition 3.5 *Let I and J be infinite-valued interpretations. We write $I \sqsubset_\infty J$, if there exists $n \in \mathbb{N}$ (that depends on I and J) such that $I \sqsubset_n J$. We write $I \sqsubseteq_\infty J$ if either $I = J$ or $I \sqsubset_\infty J$.*

It is easy to prove (see [7]) that the relation $\sqsubseteq_\infty$ on the set of infinite-valued interpretations is a partial order, while for every $n \in \mathbb{N}$, the relation $\sqsubseteq_n$ is a preorder. Moreover, the following theorem holds [7]:

Theorem 3.6 *Every normal propositional logic program P has a unique minimum infinite-valued model M_P under $\sqsubseteq_\infty$.*

The above theorem establishes a syntax-independent characterization of the semantics of logic programs with negation. Moreover, the following theorem of [7] demonstrates that the infinite-valued model is compatible with the well-founded one:

Theorem 3.7 *Let P be a normal propositional logic program and let N_P be the 3-valued interpretation that results from the minimum infinite-valued model M_P of P by collapsing all the T_i values to $True$ and all the F_i values to $False$. Then, N_P is the well-founded model of P.*

The above theorem actually suggests that the infinite-valued model is a refinement of the well-founded one.

4 Strong Equivalence under the Infinite-Valued Semantics

In this section we demonstrate that strong equivalence under the infinite-valued semantics corresponds to logical equivalence in infinite-valued logic. We start with a simple lemma which shows how we can essentially encode truth values in V by program variables:

Lemma 4.1 *For every $n \in \mathbb{N}$, there exists a program P using only the propositional symbols $r_{\mathrm{F}_0}, \ldots, r_{\mathrm{F}_n}, r_{\mathrm{T}_0}, \ldots, r_{\mathrm{T}_n}, r_0$, such that $M_P(r_{\mathrm{F}_i}) = \mathrm{F}_i$, $M_P(r_{\mathrm{T}_i}) = \mathrm{T}_i$ and $M_P(r_0) = 0$, where M_P is the minimum infinite-valued model of P.*

Proof. Take P to be the program:

$$
\begin{array}{lcl}
r_0 & \leftarrow & \sim r_0 \\
r_{\mathrm{F}_n} & \leftarrow & \sim r_{\mathrm{T}_{n-1}} \\
r_{\mathrm{T}_n} & \leftarrow & \sim r_{\mathrm{F}_{n-1}} \\
 & \cdots & \\
r_{\mathrm{F}_1} & \leftarrow & \sim r_{\mathrm{T}_0} \\
r_{\mathrm{T}_1} & \leftarrow & \sim r_{\mathrm{F}_0} \\
r_{\mathrm{T}_0} & \leftarrow &
\end{array}
$$

It is then easy to see that the minimum infinite-valued model of the above program (see [7] for how this model can be constructed) satisfies the conditions specified by the lemma. ∎

Theorem 4.2 *Two normal propositional logic programs P_1, P_2 are strongly equivalent under the infinite-valued semantics if and only if they are logically equivalent in infinite-valued logic.*

PROOF. ($\Leftarrow$) Assume that P_1 and P_2 are logically equivalent in infinite-valued logic. Then, every infinite-valued model that satisfies one of them also satisfies the other. This immediately implies that for all programs P, $P_1 \cup P$ and $P_2 \cup P$ are logically equivalent in infinite-valued logic, i.e., they have the same sets of infinite-valued models. Thus, $P_1 \cup P$ and $P_2 \cup P$ also have the same minimum infinite-valued model (which exists by Theorem 3.6). Therefore, P_1 and P_2 are strongly equivalent under the infinite-valued semantics.

($\Rightarrow$) Assume that P_1 and P_2 are strongly equivalent under the infinite-valued semantics, but not logically equivalent in infinite-valued logic. Then, without loss of generality, we may assume that P_1 has a model M which is not a model of P_2; moreover, we may assume that $M(q) = \mathrm{F}_0$, for every $q \in Q$ that does not appear in $P_1 \cup P_2$. We show how a program P can be constructed, such that $P_1 \cup P$ and $P_2 \cup P$ do not have the same minimum infinite-valued model. First, P contains the rules that encode all the truth values assigned by M to the propositional symbols that appear in P_1, as demonstrated by Lemma 4.1. In other words, P contains clauses regarding the propositional symbols $r_{\mathrm{F}_0}, \ldots, r_{\mathrm{F}_n}, r_{\mathrm{T}_0}, \ldots, r_{\mathrm{T}_n}, r_0$ (disjoint from the propositional symbols that appear in $P_1 \cup P_2$), where n is the largest subscript of a truth value assigned by M to propositional symbols of P_1. Moreover, for every propositional symbol p in P_1, P contains the clause $p \leftarrow r_{M(p)}$. Let N be an interpretation identical to M the only difference being that N additionally assigns to each propositional symbol r_v the appropriate value, i.e., $N(r_v) = v$.

We claim that N is the minimum infinite-valued model of $P_1 \cup P$. Obviously, N is a model of $P_1 \cup P$ since it is a model of both P_1 and P. Now, assume there exists N^* such that $N^* \sqsubset_\infty N$ and N^* is a model of $P_1 \cup P$. Then, there exists $k \in \mathbb{N}$ such that $N^* \sqsubset_k N$. This means that there exists a propositional symbol p that appears in P_1 such that either $N^*(p) = \mathrm{F}_k$ and $N(p) > \mathrm{F}_k$ or $N(p) = \mathrm{T}_k$ and $N^*(p) < \mathrm{T}_k$. But then, in both cases, $N^*(p) < N(p) = M(p)$, which implies that N^* does not satisfy the clause $p \leftarrow r_{M(p)}$ that exists in P. In other words, N^* is not a model of P and consequently not a model of $P_1 \cup P$ (contradiction). Therefore, N is the minimum infinite-valued model of $P_1 \cup P$ and since P_1 and P_2 are strongly equivalent, N is also the minimum infinite-valued model of $P_2 \cup P$. But then, N is also a model of P_2 which also implies that M is a model of P_2 (contradiction). ∎

The above theorem leads to a sufficient condition for strong equivalence under the well-founded semantics:

Corollary 4.3 *If two normal propositional logic programs P_1 and P_2 are logically equivalent in infinite-valued logic then they are strongly equivalent under the well-founded semantics.*

PROOF. It follows immediately by combining Theorems 4.2 and 3.7. ∎

It should be noted here that the study of strong equivalence under the well-founded semantics, is much more limited than that for the answer set semantics. The only other relevant work we are aware of, is the one reported in [1].

Example 4.4 *Consider the programs:*

$$\begin{array}{lcl} p \leftarrow \sim q \\ q \leftarrow r \\ r \leftarrow q \end{array} \qquad \text{and} \qquad \begin{array}{lcl} p \leftarrow \sim r \\ q \leftarrow r \\ r \leftarrow q \end{array}$$

Notice now that given a model M of the first program, we must have $M(q) = M(r)$, and similarly for any model of the second program (due to the second and third clauses of the programs). This immediately leads to the fact that the two programs are logically equivalent and therefore strongly equivalent under the infinite-valued semantics. By Corollary 4.3, the two programs are strongly equivalent under the well-founded semantics.

5 Complexity of Strong Equivalence

In this section we examine the complexity of deciding whether two given propositional normal logic programs are strongly equivalent under the infinite-valued semantics. Since the underlying logic has an infinite number of truth values, it is not an obvious fact that the above problem is decidable. In order to obtain the decidability result, we prove that if two programs are not logically equivalent in the infinite-valued logic, then there exists an interpretation using truth values with small indices that witnesses this fact. A more careful inspection shows that the problem is actually in **coNP**. We also prove that the problem is **coNP**-hard, using a reduction form 3SAT to its complementary problem. The above **coNP**-completeness result means that if two programs are not strongly equivalent, then there exists a short (with respect to the size of the programs) certificate, which can be efficiently checked. However, it is unlikely that we can effectively construct this certificate for a given pair of programs in polynomial time. Moreover, it is unlikely that a similar certificate exists if two programs are logically equivalent.

Theorem 5.1 *The problem of whether two normal propositional logic programs are strongly equivalent under the infinite-valued semantics is in* **coNP**.

PROOF. Let P_1, P_2 be two programs that are not strongly equivalent and let S be the set of propositional symbols that appear in $P_1 \cup P_2$. By Theorem 4.2, without loss of generality, there exists an interpretation I that is a model of P_1, but not a model of P_2. We will first prove that there exists an interpretation J, such that for every $p \in S$, $J(p) \in \{0, \mathrm{F}_0, \mathrm{T}_0, \mathrm{F}_1, \mathrm{T}_1, \ldots, \mathrm{F}_{2\cdot|S|}, \mathrm{T}_{2\cdot|S|}\}$, which is also a model of P_1, but not a model of P_2.

Let L be the set of all literals constructed from propositional symbols in S. We define the function $r : \mathbb{N} \to \mathbb{N}$ as follows:

$$r(n) = |\{k : k < n \text{ and } \exists \ell \in L \text{ such that } I(\ell) = \mathrm{T}_k \text{ or } I(\ell) = \mathrm{F}_k\}|$$

Notice that r is a non-decreasing function satisfying the property $r(n) \leq |L| = 2 \cdot |S|$ for every $n \in \mathbb{N}$. Moreover, if there exists some $\ell \in L$ such that $I(\ell) \in \{\mathrm{T}_i, \mathrm{F}_i\}$, then $r(i+1) = r(i) + 1$.

Define the following mapping between truth values:

$$h(v) = \begin{cases} \mathrm{T}_{r(n)} & \text{if } v = \mathrm{T}_n \\ \mathrm{F}_{r(n)} & \text{if } v = \mathrm{F}_n \\ 0 & \text{if } v = 0 \end{cases}$$

It is easy to see that h is an increasing function (i.e., if $v_1 \leq v_2$ then $h(v_1) \leq h(v_2)$). Furthermore, the restriction of h to the set $V^* = \{v \in V \mid \exists \ell \in L \text{ such that } I(\ell) = v\}$, which contains the truth values assigned to literals in L by I, is a strictly increasing function (i.e., if $v_1 < v_2$ then $h(v_1) < h(v_2)$).

Let $J(p) = h(I(p))$. Since $r(n) \leq 2{\cdot}|S|$, it holds that $J(p) \in \{0, \mathrm{F}_0, \mathrm{T}_0, \mathrm{F}_1, \mathrm{T}_1, \ldots, \mathrm{F}_{2\cdot|S|}, \mathrm{T}_{2\cdot|S|}\}$, for every $p \in S$. We will next show that J satisfies a clause $C \in (P_1 \cup P_2)$, if and only if I satisfies C.

Consider first a negative literal $\sim p$. If $I(p) = \mathrm{T}_i$ then $J(p) = \mathrm{T}_{r(i)}$ and $J(\sim p) = \mathrm{F}_{r(i)+1} = \mathrm{F}_{r(i+1)} = h(\mathrm{F}_{i+1}) = h(I(\sim p))$. Similarly, if $I(p) = \mathrm{F}_i$ then $J(\sim p) = h(\mathrm{T}_{i+1}) = h(I(\sim p))$ and if $I(p) = 0$ then $J(\sim p) = 0 = h(I(\sim p))$. Therefore, $J(\ell) = h(I(\ell))$, for every $\ell \in L$.

Now consider a clause $C = p \leftarrow \ell_1, \ell_2, \ldots, \ell_n$ in $P_1 \cup P_2$. If I satisfies C then $I(p) \geq I(\ell_i)$, for some i, $1 \leq i \leq n$. This implies that $J(p) = h(I(p)) \geq h(I(\ell_i)) = J(\ell_i)$, for some i, $1 \leq i \leq n$ (using the fact that h is increasing). Therefore, J also satisfies C.

Conversely, if I does not satisfy C then $I(p) < I(\ell_i)$, for every i, $1 \leq i \leq n$. This implies that $J(p) = h(I(p)) < h(I(\ell_i)) = J(\ell_i)$, for every i, $1 \leq i \leq n$, (using the fact that the restriction of h to V^* is strictly increasing). Consequently, J does not satisfy C.

Therefore, J satisfies exactly the same clauses of $P_1 \cup P_2$ as I, which implies that J is a model of P_1, but not a model of P_2. Moreover, every truth value used by J can be represented using $\mathcal{O}(\log |S|)$ symbols; thus, J can be represented by a string of length polynomial to the total size of P_1 and P_2. Finally, if the string representing J is given, we can verify in polynomial time that it is actually a model of P_1, but not a model of P_2.

Consequently, every pair of programs that are not strongly equivalent (that is, every 'no'-instance of the problem) possesses a polynomial certificate, which can be verified in polynomial time. Thus, deciding whether two programs are strongly equivalent is in **coNP**. ∎

Example 5.2 *Consider the programs* $\{q \leftarrow q, \sim q, \quad q \leftarrow\sim q\}$ *and* $\{q \leftarrow q, \quad q \leftarrow\sim q\}$. *In order to demonstrate that they are strongly equivalent it suffices to demonstrate that they are equivalent for all interpretations* I *such that* $I(q) \in \{\mathrm{F}_0, \mathrm{F}_1, \mathrm{F}_2, 0, \mathrm{T}_2, \mathrm{T}_1, \mathrm{T}_0\}$. *By inspection, it follows easily that the two programs are indeed strongly equivalent.*

Theorem 5.3 *The problem of whether two normal propositional logic programs are strongly equivalent under the infinite-valued semantics is* **coNP***-complete.*

Proof. By Theorem 5.1 the problem is in **coNP**. In order to prove that it is **coNP**-hard, it suffices to prove that its complementary problem is **NP**-hard. We will prove this fact using a polynomial-time reduction from 3SAT.

Let ϕ be an instance of 3SAT. That is, $\phi = \bigwedge_{i=1}^{n} c_i$, where $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ and $\ell_{i,j}$ is either a variable or the negation of a variable. Let $v_1, v_2, \ldots, v_m$ be all the variables that appear in ϕ (we may assume that these variables are elements of Q) and let p, q be two other variables from Q. We construct the following sets of rules: $A = \{p \leftarrow q, \sim q\}$; $B = \{B_1, B_2, \ldots, B_m\}$, where B_k is $p \leftarrow v_k, \sim v_k$; and $C = \{C_1, C_2, \ldots, C_n\}$, where C_i is $p \leftarrow \ell_{i,1}, \ell_{i,2}, \ell_{i,3}$. Let $P_1 = A \cup B \cup C$ and $P_2 = B \cup C$. Obviously this construction can be performed in polynomial time. We will prove that ϕ is satisfiable iff P_1 and P_2 are not strongly equivalent.

Assume that ϕ is satisfiable. That is, there exists a 2-valued truth assignment s such that $s(\phi) = True$. Consider the infinite-valued interpretation I defined as follows:

$$I(v_k) = \mathrm{T}_0 \text{ iff } s(v_k) = False$$

$$I(v_k) = \mathrm{F}_0 \text{ iff } s(v_k) = True$$

$$I(q) = 0$$

$$I(p) = \mathrm{F}_1$$
$$I(r) = \mathrm{F}_0 \text{ for all other } r \in Q$$

Since one of v_k, $\sim v_k$ has value F_0 or F_1, I satisfies every clause B_k in B. Moreover, since s satisfies ϕ, every clause c_i in ϕ contains a literal $\ell_{i,j}$ with $s(\ell_{i,j}) = True$, which implies that $I(\ell_{i,j})$ is either F_0 or F_1. Consequently, I satisfies every clause C_i in C. On the other hand I does not satisfy the clause in A. Therefore, I is a model of P_2, but not a model of P_1 and by Theorem 4.2, P_1 and P_2 are not strongly equivalent.

Conversely, assume that P_1 and P_2 are not strongly equivalent. By Theorem 4.2, since $P_2 \subset P_1$, there exists an infinite-valued interpretation I that is a model of P_2 but not a model of P_1, i.e., it satisfies the clauses in $B \cup C$ but not the clause in A. I has the following properties:

1. $I(p) < 0$, since I does not satisfy the clause in A.
2. $I(v_k) \neq 0$ for all k, since I satisfies B_k and property 1 holds.

Consider the 2-valued truth assignment s with $s(v_k) = True$ iff $I(v_k) < 0$. We claim that s satisfies ϕ. To prove this claim, consider any clause c_i in ϕ. Since I satisfies C_i and $I(p) < 0$, there exists j such that $I(\ell_{i,j}) < 0$. Thus $s(\ell_{i,j}) = True$, which implies $s(c_i) = True$. Consequently, $s(\phi) = True$, which proves that ϕ is satisfiable. ∎

6 Discussion

The work presented in this paper is the first to examine the problem of strong equivalence under the infinite-valued semantics for negation. Our results demonstrate that strong equivalence under this semantics coincides with logical equivalence in the corresponding infinite-valued logic. On the negative side, we have demonstrated that the problem of testing two programs for strong equivalence, is coNP-complete. This implies that one can not expect in general to devise efficient algorithms that can check for strong equivalence. Actually, for more extended languages (eg., first-order logic programs), testing for any sensible form of program equivalence, becomes undecidable. What is then the benefit of studying such notions that may have an unmanageable complexity? Strong equivalence (as-well-as other related notions) are very useful in that they offer us a framework under which one can define and verify various program transformation techniques. Notice that, in designing program transformation techniques, there is no need to decide about the strong equivalence of two arbitrary programs, which might be intractable. Instead, one seeks for transformation schemes which can be *performed efficiently* and *preserve strong equivalence*. This is very similar to a situation that also arises in logic: although deciding the satisfiability, validity or equivalence of arbitrary propositional formulas is intractable, we can efficiently construct satisfiable or valid formulas or transform a given formula to an equivalent one using well-known properties of boolean connectives. In conclusion, defining transformations that preserve strong equivalence is very important since they can then be used in order to preprocess and optimize a program before actual execution. Defining such transformations is a worthy program that needs further investigation (which is beyond the scope of this paper).

Closing, we believe that the infinite-valued semantics is a promising approach for further semantic investigations in the theory of logic programming. Our belief is based (for example) on the recent, purely game-theoretic approach to the semantics of negation in logic programming [2], which was motivated and proved correct based on the infinite-valued approach. A

natural question for future work would be to consider the notion of strong equivalence for generalized disjunctive logic programs under the infinite-valued semantics (or for even more extended classes of programs). We are currently investigating such issues.

References

[1] P. Cabalar, S. P. Odintsov and D. Pearce. Logical Foundations of Well-Founded Semantics. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 25-35, 2006.

[2] Ch. Galanaki, P. Rondogiannis and W.W. Wadge. An Infinite-Game Semantics for Well-Founded Negation in Logic Programming. *Annals of Pure and Applied Logic*, 151(2–3):70–88, 2008.

[3] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080. MIT Press, 1988.

[4] V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[5] Ch. Nomikos, P. Rondogiannis, and W. W. Wadge. A Sufficient Condition for Strong Equivalence Under the Well-Founded Semantics. In Proceedings of the 21st International Conference on Logic Programming, pages 414–415, 2005.

[6] D. Pearce. A new Logical Characterization of Stable Models and Answer Sets. In L. Pereira J. Dix and T. Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming*, pages 57–70. Springer-Verlag, 1997.

[7] P. Rondogiannis and W.W. Wadge. Minimum Model Semantics for Logic Programs with Negation-as-Failure. *ACM Transactions on Computational Logic*, 6(2): 441-467, 2005.

[8] A. van Gelder, K. A. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.